Realizing Reconfigurable Mesh Algorithms on Softcore Arrays

Heiner Giefers, Marco Platzner
Computer Engineering Group
University of Paderborn

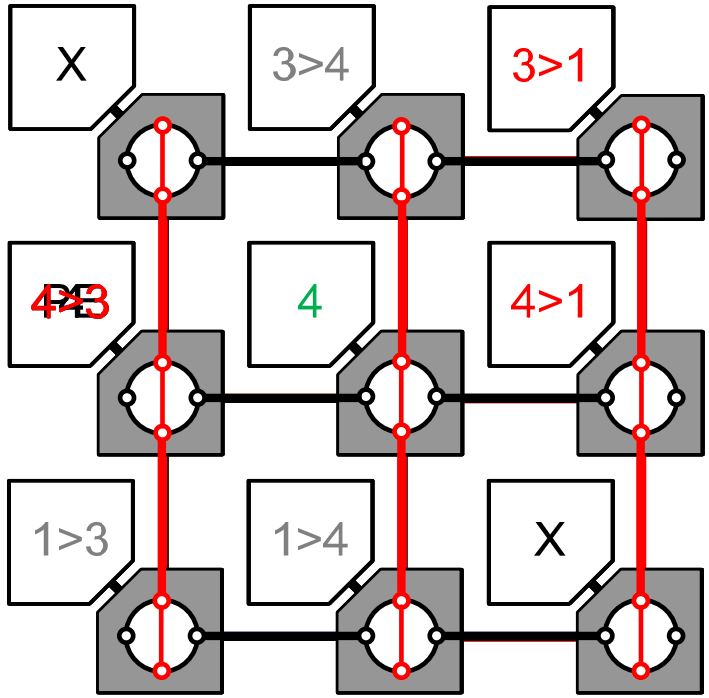
{hgiefers, platzner}@upb.de

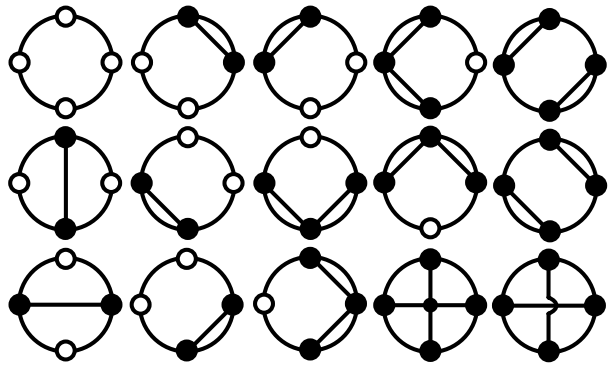
Outline

1. Reconfigurable mesh model
2. Implementing reconfigurable meshes on softcore arrays
3. Execution of reconfigurable mesh algorithms
4. Case studies
5. Summary and outlook

Reconfigurable Mesh Model (R-Mesh)

- Cores execute in a lock step
  1. Switch configuration
  2. Communication
  3. (Constant time) computation
- Connection autonomy
  - 15 possible patterns
- Many model extensions
- Communication delay model
  - O(*1*) and O(*log(N)*)
- Example: MAX
  - Complexity: O(*1*)
  - Communication distance: *3N*





Algorithm Mapping

- Basic algorithms
  - Arithmetic [Miller et al. 93]
    - Prefix sum of N elements, O(*1*) on N x N mesh
    - Minimum of N elements, O(*1*) on N x N mesh
  - Sorting of N elements
    - On N x N mesh in constant time [Jang and Prasanna 95]
    - On $N^{1/2}$ x $N^{1/2}$ mesh in O($N^{1/2}$) [Thompson and Kung 77]
  - Graph algorithms
  - Image processing

- Simulation of other network topologies
  - Pyramid, simulation overhead O(*1*)
  - Mesh-of-Trees, simulation overhead O(*log(N)*)

Practical Issues

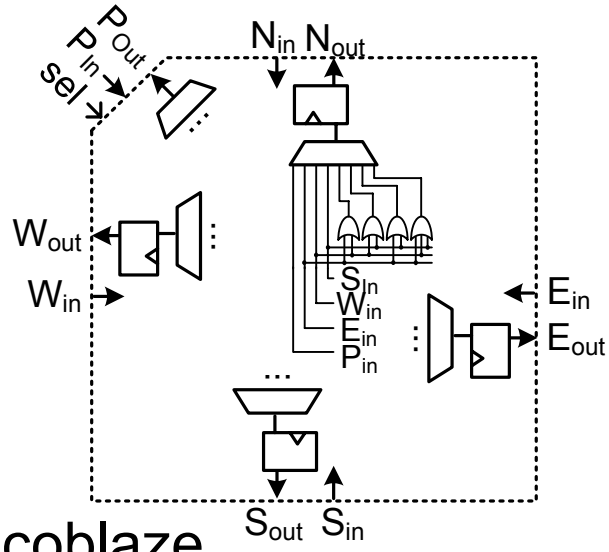
- Advantages of the R-Mesh model
  - Scalable parallel programming model
  - Algorithms for numerous problems with outstanding runtime complexities
- Challenges of an R-Mesh implementation
  - Communication delays of *O(1)* or *O(log N)* are unrealistic
  - Data I/O is not covered by runtime analysis
- Our goal: reduce the R-Mesh model to practice
  - (How) Can we exploit the theoretical quality while accepting multi-cycle communication?

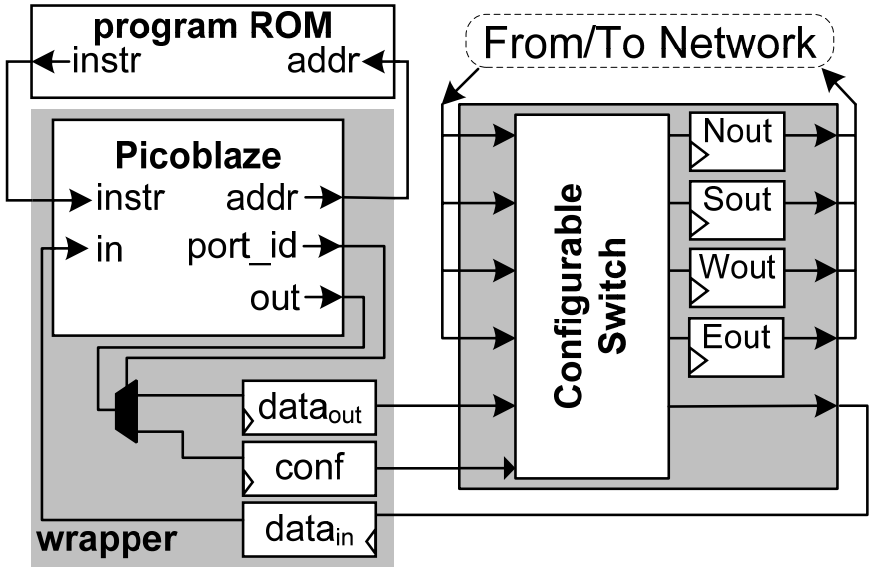
Outline

1. Reconfigurable mesh model ✓
2. Implementing reconfigurable meshes on softcore arrays
3. Execution of reconfigurable mesh algorithms
4. Case studies
5. Summary and outlook

Building Blocks: Switch & Processing Element

- Reconfigurable switch
  1. Combinatorial switch [FPL07]
  2. Buffered switch
  3. *Self timed switch*
  4. *Photonic switch*

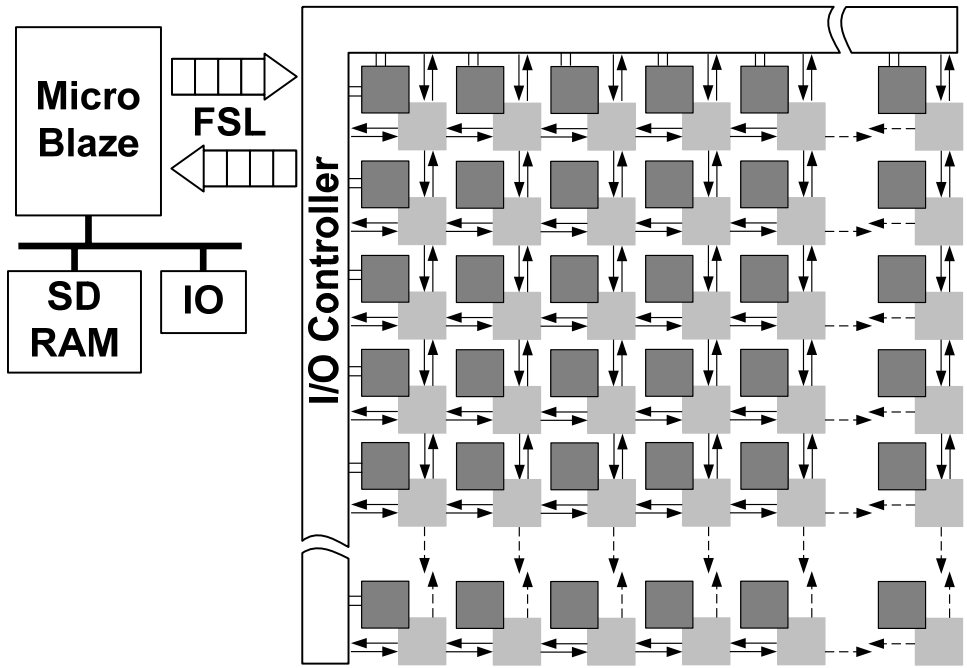




- Picoblaze
  - 8-bit RISC, not pipelined
  - 96 slices on Virtex-II
  - 1024 instructions in 1 BRAM
  - 2 cycles per instruction
  - 64 byte scratchpad RAM
- Wrapper
  - Storing data and switch configuration
  - Optional I/O functionality

Host Integration

- RMESH as coprocessor
- MicroBlaze 32-bit RISC core as host CPU
- Connected through FSL channels to an I/O controller
  - Low latency of FSL
  - 32-bit to 4 x 8-bit splitting
  - Load and store operations take O(*N*) time
- Entire system runs at 100 MHz



Synthesis Results for Virtex 4 (XCV4LX200)

	slices (utilization)	BRAM	f_max [MHz]
PE1: pBlaze + wrapper	114 (0.13%)	1	184
PE2: pBlaze + wrapper + multiplier	131 (0.15%)	1	184
SE1: RMESH switch	144 (0.15%)	0	192
SE2: HVRM switch	67 (0.08%)	0	237
Node1: PE1+SE1	265 (0.30%)	1	156
Node2: PE1+SE2	178 (0.20%)	1	156
Node3: PE2+SE1	282 (0.32%)	1	156
Node4: PE2+SE2	195 (0.22%)	1	156

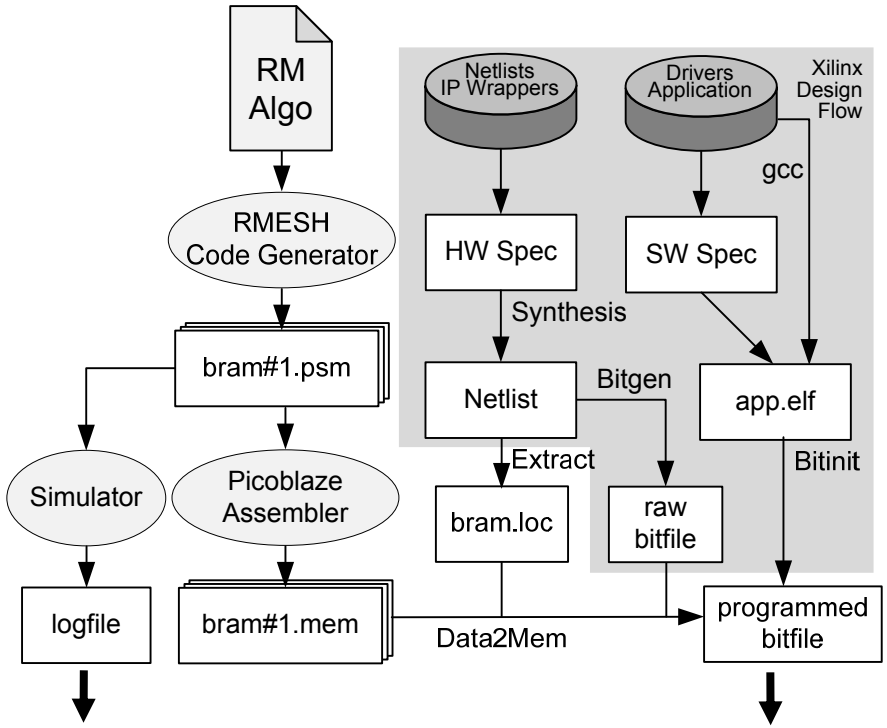
Building Blocks

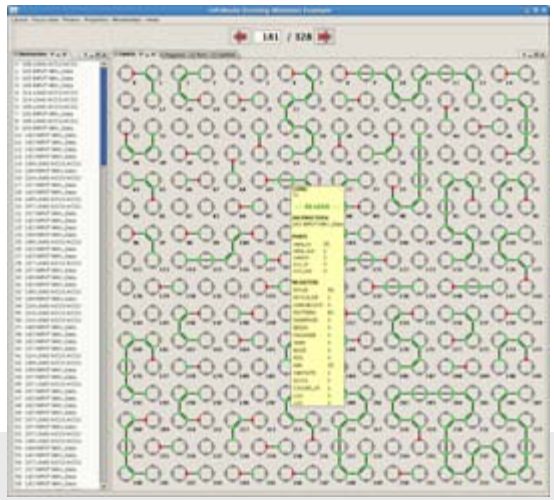
Network Tiles

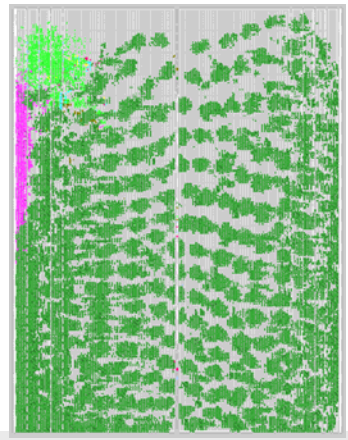
- 16x16 Prototype: µBlaze, FSLs, 256 Node1 (PE1+SE1)
  - 65091 slices (73%)
  - 264 BRAMs (78%)
  - 100 MHz

Design Tool Flow

- Xilinx EDK Flow
  - HW generation
  - µBlaze programming
- R-Mesh algorithm
- Assembly code generator
- Picoblaze assembler
- Bitfile modification
- Cycle-accurate simulator
  - Simulate arbitrary mesh sizes







Outline

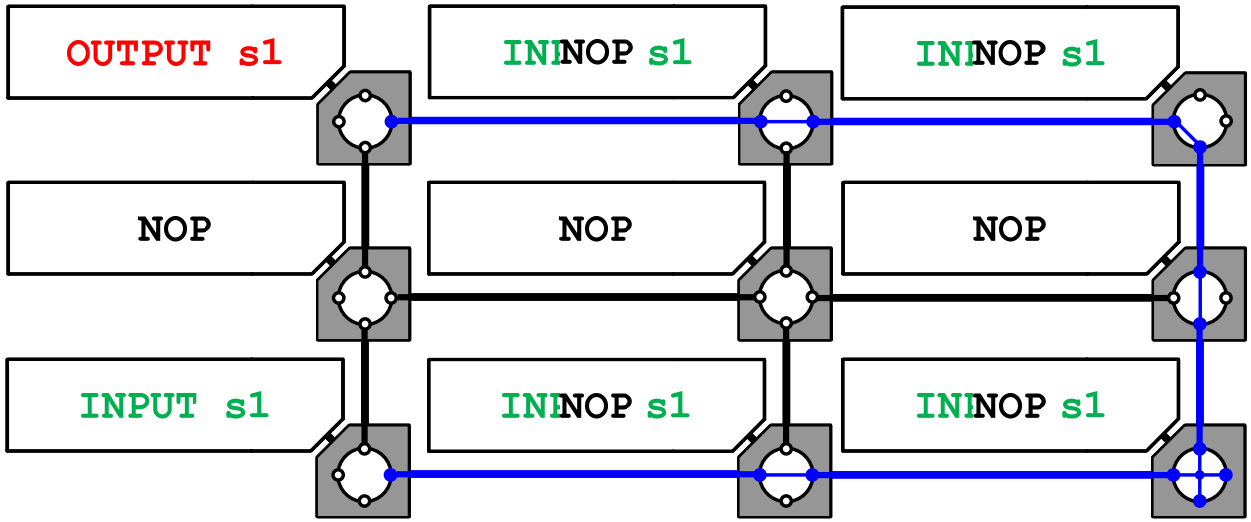
1. Reconfigurable mesh model ✓
2. Implementing reconfigurable meshes on softcore arrays ✓
3. Execution of reconfigurable mesh algorithms
4. Case studies
5. Summary and outlook

Reconfigurable Mesh Execution

- R-Mesh requirements
  - Cores execute in a lock step
    1. Switch configuration
    2. *low latency* communication
    3. *bounded* computation
  - Communication must be synchronized
- Observations
  - Communication pattern often independent of static data, thus known at compile time
  - Complexity of computation phase varies
- Implementation principles
  - Very low communication overhead
    - Configure the switch with an atomic instruction
    - Write registers directly to the mesh
  - Select algorithms that match our architecture
    - Keep linear communication overhead in mind
    - Increase computation/communication ratio

Synchronisation

1. Commmunication in a single (clock) cycle
   - Inappropriate for large scale meshes
2. Combinatoric switches with multi-cycle paths
   - Tedious synthesis process (at least with Xilinx place&route tools)
3. Buffered switches
   - Extend data with valid bit
   - More robust and flexible



Algorithm Selection

- Complexity Analysis bases on constant time delay
  - Algorithms with best computational complexity usually make heavy use of broadcasts
  - Our architecture profits from short distance communication
  - ➔ Select / develop proper algorithms
- Example: 3 maximum finding algorithms
  1. Classical R-Mesh algorithm with constant runtime
  2. Computes the Maximum of N values on an R-Mesh of size N by sequentially comparing all k bits of all values
  3. Simulates a binary tree algorithm for maximum finding on R-Mesh

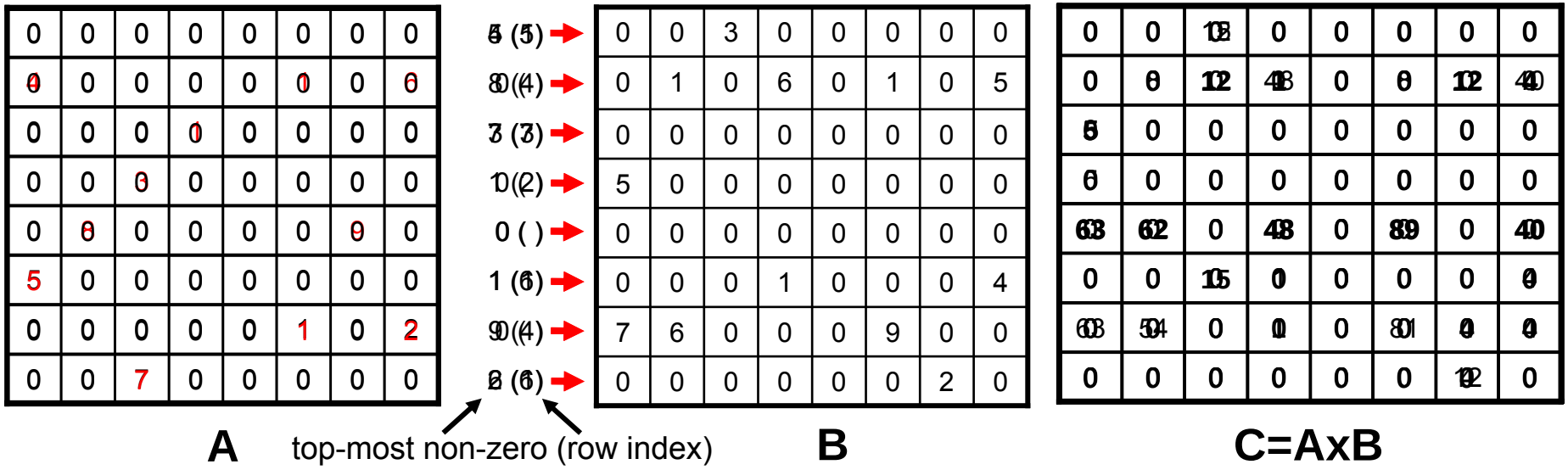
Algorithm	Datasize	Area	Complexity	Communication
#1	*N*	*N x N*	O(*1*)	*3N* [cycles]
#2	*N*	*N*	O(*k*)	$2kN^{½}$ [cycles]
#3	*N*	*N*	O(*log*($N^{½}$))	$2N^{½}$ [cycles]

Outline

1. Reconfigurable mesh model ✓
2. Implementing reconfigurable meshes on softcore arrays ✓
3. Execution of reconfigurable mesh algorithms ✓
4. Case studies
5. Summary and outlook

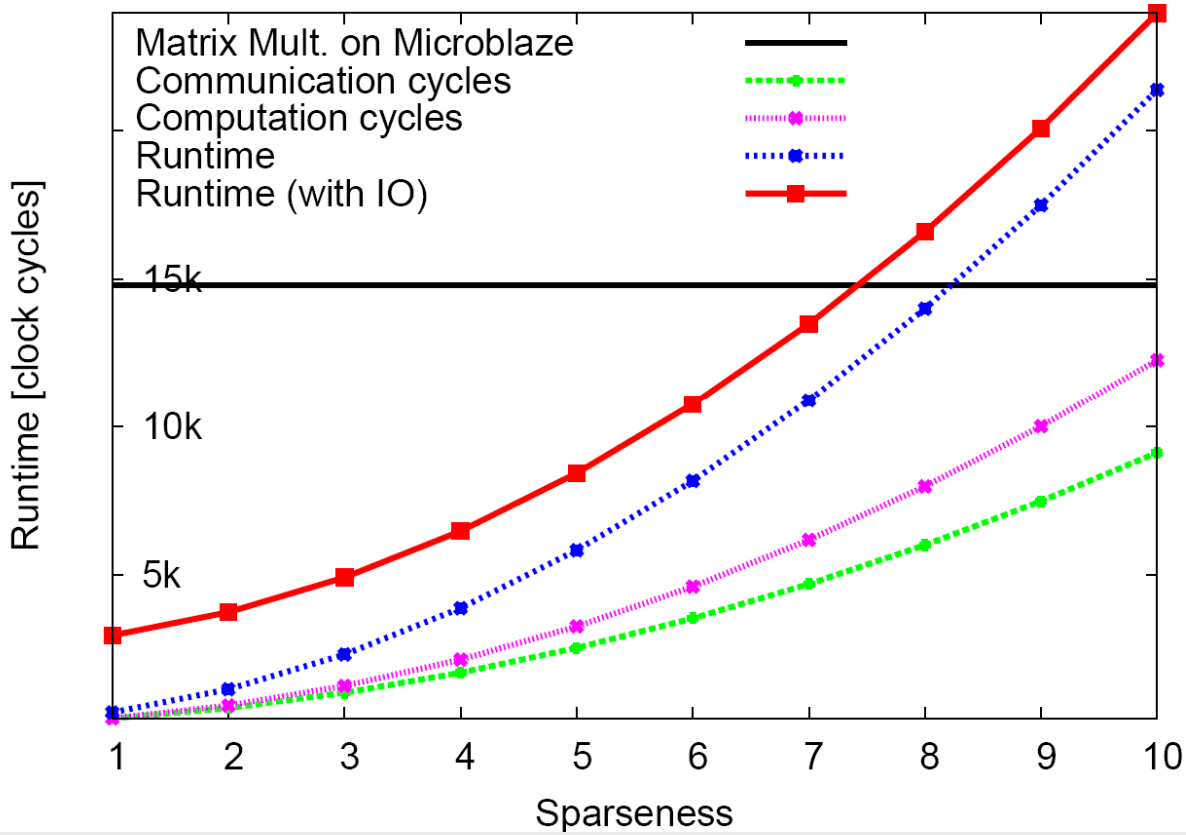
Case Study: Sparse Matrix Multiplication

- RMESH algorithm for column/row sparse matrix multiplication [Middendorf et al. 99]
- Column sparse (≤ k non-zero Elements/column) Algorithm:
  - Repeat (at most k(=2) times)
    - Broadcast the top-most non-zero elements of A to the appropriate rows of B
    - Multiply the received A-element with the local B-element
    - Route the Products to the row index given by the A-element (≤ k steps)
    - Discard the top-most A-element



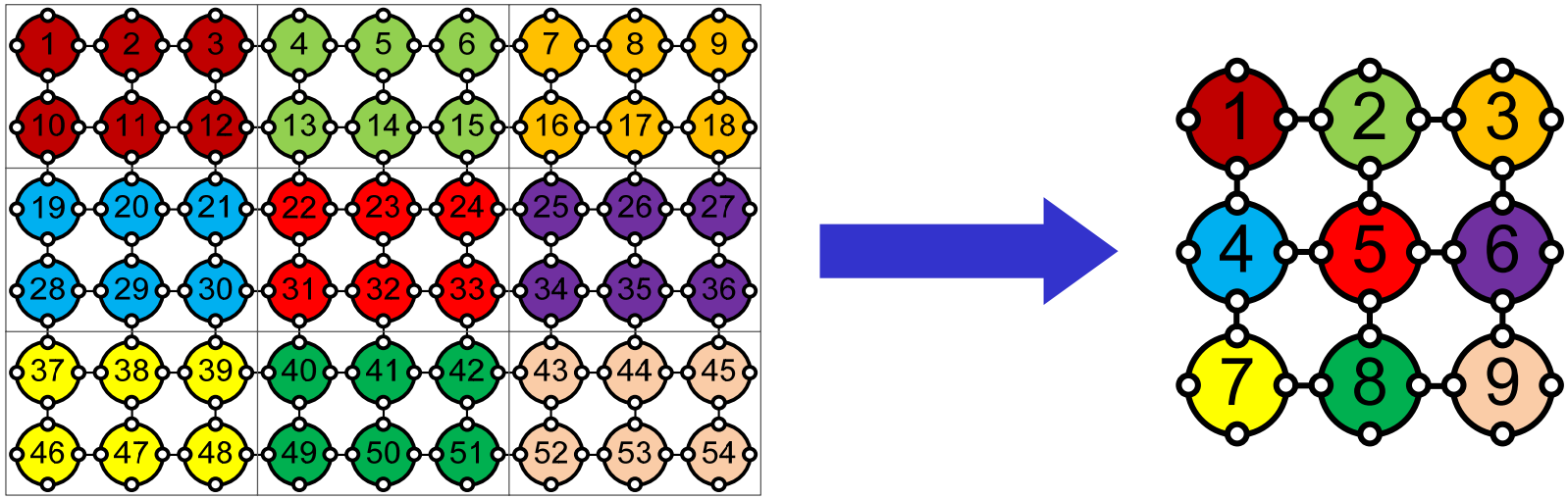
Runtime Results

- Multiplication of 2 16x16 sparse matrices on a 16x16 mesh
  - Algorithm complexity is $O(k^2)$
  - Matrices are fully loaded → constant I/O overhead of 2530 cycles
  - (Straight-forward) matrix multiplication on Microblaze takes 14788 cycles
  - Preloaded data and *k=1* → Speedup 74 (*k=2* → Speedup 11.5)



Scalability through Self Simulation

- Simulate an RM algorithm for *N* nodes on an RM instance (of the same model) consisting of *P* nodes (normally: P<N)
  - Simulation is dependent on RM model, not on the simulated algorithm
  - Self simulation slowdown O(*f*N/P*), with simulation overhead *f*
    - *f*=O(*1*) → optimal self simulation for HV-RM, L-RM
    - *f*=O(*P*) → strong self simulation for FR-RM (`{NSWE}` and `{NS,WE}` only)
    - *f*=O(*log N*) → weak self simulation for general RM
- HV-RM simulation uses contraction mapping
  - Shortens communication distances



Self Simulation Example

- Add up *64x64 (=4096)* elements
  - Assume preloaded data

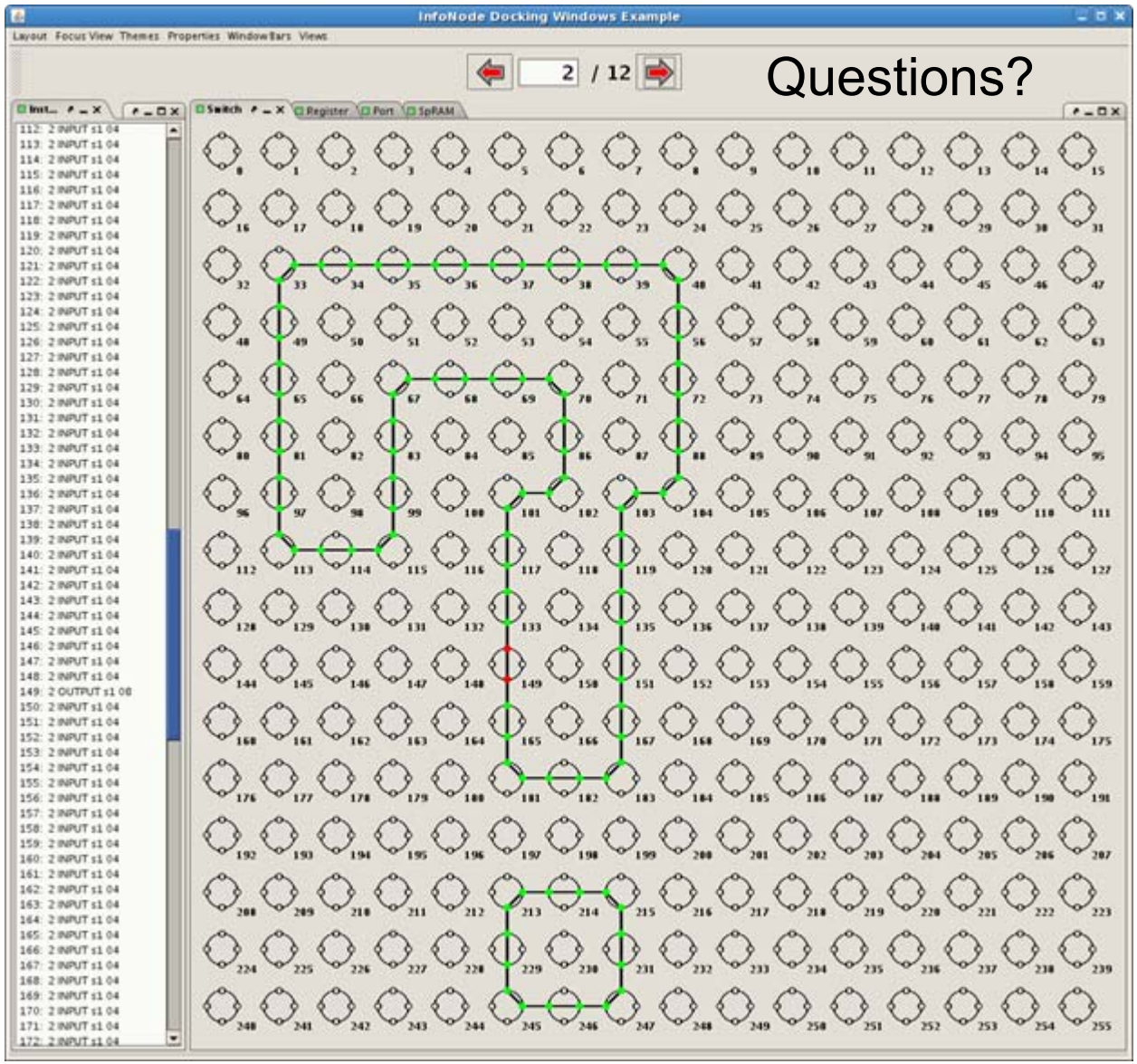
1. On single Microblaze (best Optimization): 24696 cycles
2. On 64x64 RMESH: 528 cycles
   - Speedup 46.8
3. On 16x16 RMESH: 308 cycles
   - Each node ‘simulates’ a *4x4* submesh
   - Speedup 80.2

- Self simulation can gain speedup!

	Datasize	**Runtime [cycles]**	**Speedup**
Microblaze	2^{14}	24696	1
64x64 R-Mesh	2^{14}	528	46.8
16x16 R-Mesh	2^{14}	308	80.2

Summary & Future Work

- Summary
  - Implementation of reconfigurable mesh algorithms on FPGA
  - Synchronization methods
  - Case study on sparse matrix multiplication
  - Scaling by self simulation

- Ongoing work includes
  - Improvement of the RMESH compilation flow
  - Integration of self simulation methods into the compiler
  - Analysis of our softcore array under energy aspects

Thanks for your attention!